arXiv:1806.08568v3 [cs.LG] 22 Jan 2019

Continuous Learning in Single-Incremental-Task Scenarios

Davide Maltoni & **Vincenzo Lomonaco**
Department of Computer Science and Engineering,
University of Bologna,
Via Zamboni, 33, 40126 Bologna BO, Italy
{davide.maltoni, vincenzo.lomonaco}@unibo.it

Abstract

It was recently shown that architectural, regularization and rehearsal strategies can be used to train deep models sequentially on a number of disjoint tasks without forgetting previously acquired knowledge. However, these strategies are still unsatisfactory if the tasks are not disjoint but constitute a single incremental task (e.g., class-incremental learning). In this paper we point out the differences between multi-task and single-incremental-task scenarios and show that well-known approaches such as LWF, EWC and SI are not ideal for incremental task scenarios. A new approach, denoted as AR1, combining architectural and regularization strategies is then specifically proposed. AR1 overhead (in terms of memory and computation) is very small thus making it suitable for online learning. When tested on CORe50 and iCIFAR-100, AR1 outperformed existing regularization strategies by a good margin.

1 Introduction

During the past few years we have witnessed a renewed and growing attention to Continuous Learning (CL) [3][4][25]. The interest in CL it essentially twofold. From the artificial intelligence perspective CL can be seen as another important step towards the grand goal of creating autonomous agents which can learn continuously and acquire new complex skills and knowledge. From a more practical perspective CL looks particularly appealing because it enables two important properties: adaptability and scalability. One of the key hallmarks of CL techniques is the ability to update the models by using only recent data (i.e., without accessing old data). This is often the only practical solution when learning on the edge from high-dimensional streaming or ephemeral data, which would be impossible to keep in memory and process from scratch every time a new piece of information becomes available. Unfortunately, when (deep or shallow) neural networks are trained only on new data, they experience a rapid overriding of their weights with a phenomenon known in literature as *catastrophic forgetting* [23][27][2]. Despite a number of strategies have been recently proposed to mitigate its disruptive effects, catastrophic forgetting still constitutes the main obstacle toward effective CL implementations.

1.1 Continuous Learning Strategies

The sudden interest in CL and its applications, especially in the context of deep architectures, has led to significant progress and original research directions, yet leaving the research community without a common terminology and clear objectives. Here we propose, in line with [12] and [33], a three-way fuzzy categorization of the most common CL strategies:

- **Architectural strategies**: specific architectures, layers, activation functions, and/or weight-freezing strategies are used to mitigate forgetting. It includes dual-memory models attempting to imitate hippocampus-cortex duality.
- **Regularization strategies**: the loss function is extended with loss terms promoting selective consolidation of the weights which are important to retain past memories. Include basic regularization techniques such as weight sparsification, dropout or early stopping.
- **Rehearsal strategies**: past information is periodically replayed to the model to strengthen connections for memories it has already learned. A simple approach is storing part of the previous training data and interleaving them with new patterns for future training. A more challenging approach is pseudo-rehearsal with generative models.

In the Venn diagram of Figure 1 we show some of the most popular CL strategies. While each category is being populated with an increasing number of novel strategies, there is a large room for yet-to-be-explored techniques especially at the intersection of the three categories.

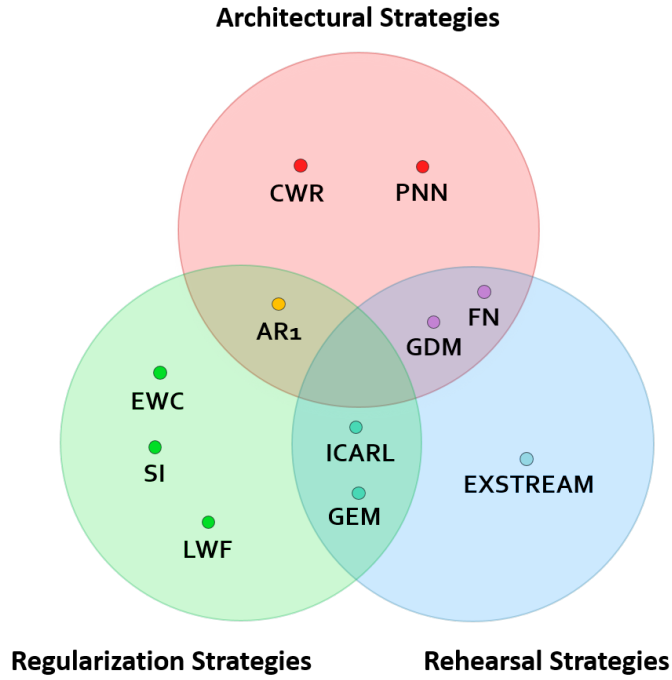

Figure 1: Venn diagram of some of the most popular CL strategies: CWR [19], PNN [29], EWC [14], SI [33], LWF [17], ICARL [28], GEM [21], FN [11], GDM [26], EXSTREAM [5] and AR1, hereby proposed. Better viewed in color.

Progressive Neural Networks (PNN) [29] is one of the first architectural strategies proposed and is based on a clever combination of parameter freezing and network expansion. While PNN was shown to be effective on short series of simple tasks, the number of the model parameters keeps increasing at least linearly with the number of tasks, making it difficult to use for long sequences. The recently proposed CopyWeights with Re-init (CWR) [19], constitutes a simpler and lighter counterpart to PNN (at the cost of a lower flexibility), with a fixed number of shared parameters and already proven to be useful on longer sequences of tasks.

Learning Without Forgetting (LWF) [17] is a regularization strategy attempting to preserve the model accuracy on old tasks by imposing output stability through knowledge distillation [8]. Other well-known regularization strategies are Elastic Weights Consolidation (EWC) and Synaptic Intelligence (SI), both articulated around a weighted quadratic regularization loss which penalizes moving weights which are important for old tasks.

At the intersection between rehearsal and regularization strategies we highlight ICARL [28] and GEM [21]. The former, whose acronym stands for *Incremental Classifier and Representation*

Learning, includes an external fixed memory to store a subset of old task data based on an elaborated sample selection procedure, but also employs a distillation step acting as a regularization; the latter, known as *Gradient Episodic Memory*, uses a fixed memory to store a subset of old patterns and add regularization constraints to the loss optimization, aimed not only at controlling forgetting but also at improving accuracy on previous tasks while learning the subsequent ones (a phenomenon known as "positive backward transfer"). A recent study on memory efficient implementation of pure rehearsal strategies is provided in [5] where a new partitioning-based method for stream clustering named EXSTREAM is shown to be very competitive with a *full rehearsal* approach (storing all the past data) and with other memory management techniques.

Very recently, a growing number of techniques have been proposed on CL based on both variations of the previously introduced strategies or completely novel approaches with different degrees of success (see [25] for a review). In particular, FearNet (FN) [11] and Growing Dual-Memory (GDM) [26] are interesting approaches leveraging ideas from architectural and (pseudo) rehearsal categories: a double-memory system is exploited to learn new concepts in a short-term memory and progressively consolidate them in a long-term one.

1.2 Continuous Learning Benchmarks

Benchmarking CL strategies today is still highly nonstandard and, even if we focus on supervised classification (e.g. leaving reinforcement learning out), researches often reports their results on different datasets by following different training and evaluation protocols (see Table 1).

Most of CL studies consider a Multi-Task (MT) scenario, where the same model is required to learn incrementally a number of isolated tasks without forgetting how to solve the previous ones. For example, in [33] MNIST is split in 5 isolated tasks, where each task consists in learning two classes (i.e. two digits). There is no class overlapping among different tasks, and accuracy is computed separately for each task. Such a model cannot be used to classify an unknown digit into the 10 classes, unless an oracle is available at inference time to associate the unknown pattern to the right task in order to setup the last classification layer(s) accordingly. In other words, these experiments are well suited for studying the feasibility of training a single model on a sequence of disjoint tasks without forgetting how to solve the previous ones, but are not appropriate for addressing incremental problems.

Table 1: Categorizations of CL experiments from the recent literature. Most of the benchmarks are based on reshaped versions of well-known datasets such as MNIST, CIFAR-10, CIFAR-100, ILSVR2012 and CUB-200. CORe50 is one of the few datasets specifically designed for CL in a SIT scenario.

Dataset	**Multi-Task (MT)**	**Single-Incremental-Task (SIT)**
Permuted MNIST	[14][21]	-
Rotated MNIST	[21]	-
MNIST Split	[33]	[25]
CIFAR-10/100 Split	[33]	-
iCIFAR-100	[21]*	[28]
ILSVRC2012	-	[28]
CUB-200	-	[25]
CORe50	-	[19]

*In [21], the authors decided to further split the iCIFAR-100 benchmark in 20 different "tasks" containing 5 different classes each.

A still largely unexplored scenario, hereafter denoted as Single-Incremental-Task (SIT) is addressed in [28] and [19]. This scenario considers a single task which is incremental in nature. An example of SIT is the so called class-incremental learning where, we still add new classes sequentially but the classification problem is unique and, when computing accuracy, we need to distinguish among all the classes encountered so far. This is quite common in natural learning, for example in object recognition, as a child learns to recognize new objects, they need to be discriminated w.r.t. the whole set of already known objects (i.e. visual recognition tasks are rarely isolated in nature).

Usually, SIT scenario is more difficult than MT one: in fact, `i)` we still have to deal with catastrophic forgetting; `ii)` we need to learn to discriminate classes that typically we never see together (e.g. in the same batch), except when a memory buffer is used to store/replay a fraction of past data.

Figure 2 graphically highlights the difference between MT and SIT. While for MT the output neurons are grouped into separate classification layers (one for each task), SIT uses a single output layer including the neurons of all the classes encountered so far. In the MT training phase, the output layer of the batch can be trained apart while sharing the rest of the model weights (denoted as $\bar{\Theta}$ in the figure). This is not the case in SIT where weights learned for the old classes could be exploited when learning the current batch classes. In the MT evaluation phase, assuming to know the task membership of each test sample, each task can be assessed separately with the corresponding classification layer. Instead, in the SIT scenario, the evaluation is performed agnostically with respect to the membership of a sample to a specific incremental batch and the final probabilities are computed through a unique softmax layer; this requires to compare objects that were never seen together during training and can have a strong impact on final accuracy. Some researchers, in the continuous learning context, use the term "head" to denote the output classification layer: using this terminology, the MT scenario can be implemented with multiple disjoint heads, while SIT is characterized by a single expanding head.

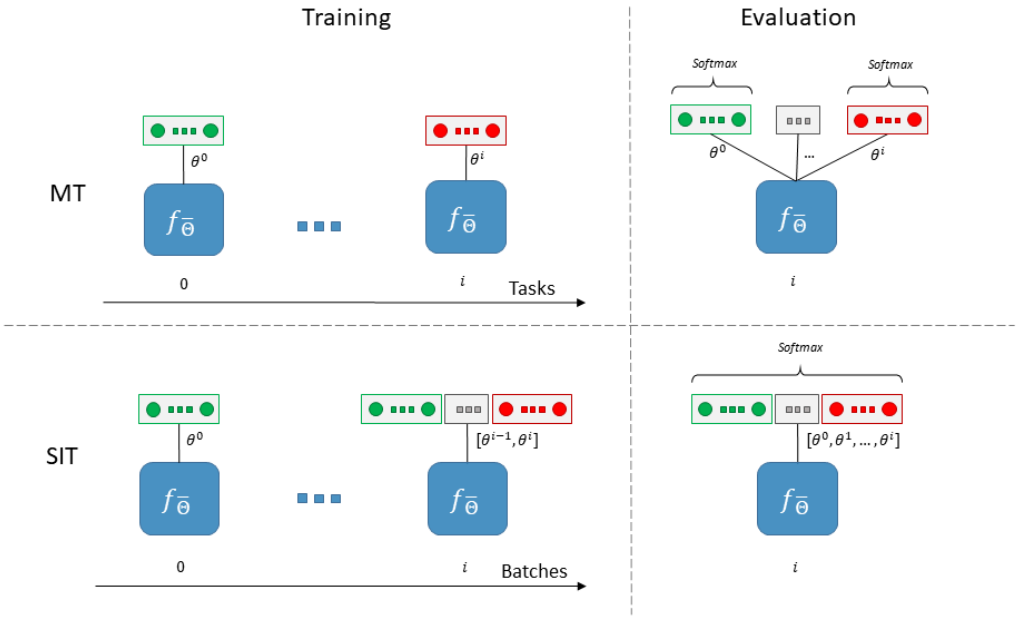

Figure 2: Key architectural differences between MT and SIT scenarios: a disjoint output layer (also denoted as "head") is used in MT for each independent task, while a single (dynamically expanded) output layer is used in SIT to include all the classes encountered so far. Better viewed in color.

Figure 3 provides an example that quantifies how much more complex SIT is than MIT on CIFAR-10/100 Split. For direct comparison with [33] here we limit the number of tasks/batches to 6 (instead of 11) and report the accuracy only at the end of training (i.e., after the 6th batch). It is evident that SIT represents a much more difficult challenge for state-of-the-art CL strategies. Looking at the average accuracy we can notice a gap between MT and SIT of more than 30% regardless the CL technique. Actually, the unbalanced nature of the CIFAR-10/100 Split benchmark (50% of all the train and test set examples belong to the first batch) makes SIT strategies quite harder to parametrize. However, as argued by other researches [11][12], most of the existing CL approaches perform well on MT (with a moderate number of tasks) but fail on complex SIT scenario with several (limited-size) batches.

Finally, it is worth noting that class-incremental learning is not the only setting of interest for SIT strategy; in [19] we introduced[1] three different update content types:

[1]In [19] NI, NC and NIC are referred to as scenarios. Since in this paper we use the terms scenario for MT and SIT, to avoid confusion we refer to NI, NC and NIC as specific update content types.

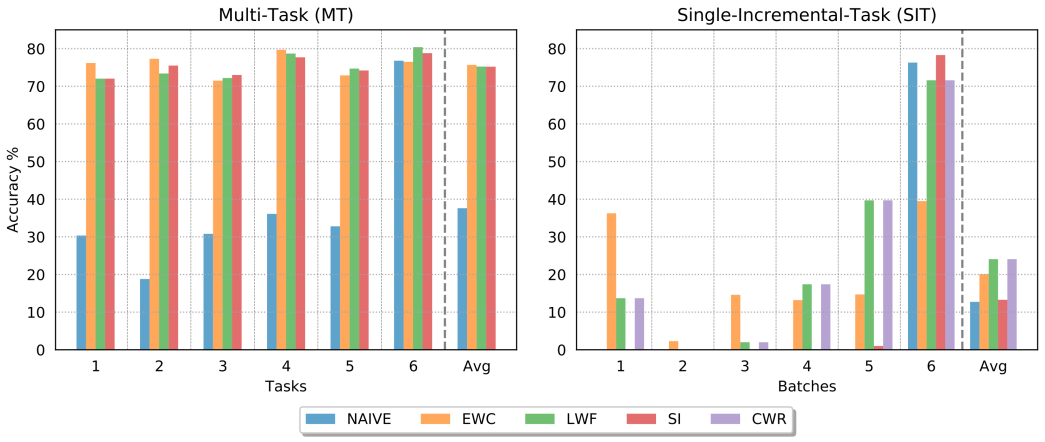

Figure 3: Accuracy results in the MT and SIT scenarios for 5 CL strategies (NAÏVE, EWC, LWF, SI, CWR) after the last training batch. With NAÏVE we denote a simple incremental fine-tuning where early stopping is the only option available to limit forgetting. Analogously to [33], this experiment was performed on the first 6 tasks of CIFAR-10/100 split. For both MT and SIT we report the accuracies on the classes of each batch $(1, 2, \ldots, 6)$ and their average (Avg). CWR is specifically designed for SIT and was not tested under MT. Better viewed in color.

- **New Instances** (NI): new training patterns of the same classes become available in subsequent batches with new poses and environment conditions (illumination, background, occlusions, etc.).
- **New Classes** (NC): new training patterns belonging to different classes become available in subsequent batches. This coincides with class-incremental learning.
- **New Instances and Classes** (NIC): new training patterns belonging to both known and new classes become available in subsequent training batches.

NI and NIC are often more appropriate than NC in real-world applications; unfortunately, MT approaches cannot operate under these settings. In the rest of this paper we focus on SIT with NC update content type since it is closer to the experiments reported so far by the research community, leaving other settings for future works. To avoid the cumbersome notation SIT-NC, hereafter we leave the second part implicit.

1.3 Contribution and Summary

The main contributions of this paper are here summarized:

- We investigate and point out the main differences between MT and SIT scenarios under the NC update content type.
- We review some popular CL approaches (LWF, EWC, SI) and the practical issues related to their implementations under SIT scenario.
- We propose a new CL strategy (AR1) by first extending the CWR architectural strategy that we introduced in [19] and then combining it with a light regularization approach such as SI.
- We compare AR1 with other common CL strategies on CORe50 and iCIFAR-100 benchmarks where we exceed by a good margin the accuracy of existing approaches.
- We discuss diagnostic techniques to simplify hyperparameters tuning in complex CL settings and to detect misbehaviors.
- In the appendices we provide implementation details of all the techniques reviewed/proposed and the experiment carried out using the Caffe framework [10].

In Section 2 we discuss the practical implementation of some popular CL strategies: in particular, we argue that ad-hoc normalization steps are necessary to make them properly work with several sequential batches under SIT scenario. For each technique we also focus on the overhead it requires in terms of computation and storage resources. In Section 3 we propose a new strategy (denoted as

AR1) which combines architectural and regularization techniques and can operate online because of a very low overhead. Section 4 and Section 5 provide experimental results and compare AR1 with existing techniques on both CORe50 and iCIFAR-100 along with some practical advices for hyperparameters tuning. Finally, in Section 6 we draw some conclusions and discuss future work.

2 Continuous Learning Strategies in SIT

In this section we review recent continuous learning techniques, and discuss their implementation in the SIT scenario. In our experience, moving from a few tasks in the MT scenario to many batches in SIT such as with CORe50, requires specific normalization techniques and careful calibration of hyper-parameters.

In this paper we focus on architectural and regularization strategies[2]: while we recognize the pragmatism and high practical value of rehearsal approaches here we concentrate on learning architectures where past experiences are "neurally" encoded; pseudo-rehearsal approaches by generative models is an emerging and very interesting approach but its practical effectiveness still needs to be proved with state-of-the-art deep architectures.

2.1 Learning Without Forgetting (LWF)

LWF [17] is a regularization approach which tries to control forgetting by imposing output (i.e. prediction) stability.

Let us consider an output level with s classes (i.e. s neurons) and assume that some classes were already learnt in previous batches. The current batch B_i includes n_i examples drawn from s_i (still unseen) classes, then LWF:

- At the beginning of batch B_i, before the training start, computes the prediction of the network for each new pattern in B_i. With this purpose it performs a forward pass and stores the s-dimensional network prediction $\hat{y}_{lwf}$ for each of the examples in B_i.
- Starts training the network with Stochastic Gradient Descent (SGD) by using a two component loss:
$$(1-\lambda) \cdot L_{cross}(\hat{y}, t=\hat{y}_{1h}) + \lambda \cdot L_{kdl}(\hat{y}, t=\hat{y}_{lwf}) \quad (1)$$
where:
  - $\hat{y}$ are the network predictions (evolving while the model is trained).
  - The first part is the usual cross-entropy loss whose target vectors t take the form of one-hot vectors $\hat{y}_{1h}$ corresponding to the true pattern labels. This component adjusts the model weights to learn the new classes in B_i.
  - The second part is a Knowledge Distillation Loss [8] which tries to keep the network predictions close to $\hat{y}_{lwf}$ (here used as soft target vectors). This component tries to preserve (for the old classes which are not in the current batch) a stable response. The second term can be replaced with one term for each old task/batch; the two formulations are equivalent but the compact form here proposed is simpler to deal with in practice.
  - The parameter $\lambda \in [0,1]$ defines the relative weights of the two loss components, thus controlling the trade-off between stability and plasticity.

In the MT scenario, L_{cross} is computed only for the n_i new classes in B_i, while L_{kdl} is computed for all the $(\sum_{j<i} n_j)$ classes previously learned. In SIT there is no such distinction and both the loss components are computed for all the s classes encountered so far.

In our LWF implementation for SIT we:

- replaced L_{kdl} with L_{cross} in the second terms. LWF authors argued in [17] that the Knowledge Distillation Loss can be replaced with Cross-Entropy with no significant accuracy change. In our initial experiments we obtained similar results, so for simplicity we adopted cross-entropy.
$$L_1 = (1-\lambda) \cdot L_{cross}(\hat{y}, t=\hat{y}_{1h}) + \lambda \cdot L_{cross}(\hat{y}, t=\hat{y}_{lwf}) \quad (2)$$

[2] Actually, in section 4.4 some preliminary experiments are included where the proposed CL strategy (AR1) is compared with well-known rehearsal techniques such as ICARL and GEM.

- fused the two loss components into a single loss with a weighted soft target vector:

$$L_2 = L_{cross}(\hat{y}, t = (1-\lambda)\cdot \hat{y}_{1h} + \lambda \cdot \hat{y}_{lwf}) \tag{3}$$

It can be simply proved that L_1 and L_2 are equivalent and lead to the same gradient flow. In fact, for cross-entropy the gradient of the loss function with respect to the logit layer o (i.e., the layer before softmax) is $\partial L_{cross}/\partial o = (\hat{y} - t)$ and therefore:

$$\begin{aligned}\frac{\partial L_1}{\partial o} = (1-\lambda)\cdot(\hat{y} - \hat{y}_{1h}) + \lambda \cdot (\hat{y} - \hat{y}_{lwf}) = \\ (\hat{y} - \hat{y}_{1h}) + \lambda \cdot (\hat{y}_{1h} - \hat{y}_{lwf}) = \\ \hat{y} - ((1-\lambda)\cdot \hat{y}_{1h} + \lambda \cdot \hat{y}_{lwf}) = \frac{\partial L_2}{\partial o}.\end{aligned} \tag{4}$$

Using a single value of λ across the sequential training batches can be suboptimal, since the importance of the past should increase with the number of classes learnt. We experimentally found[3] that a reasonable solution is increasing λ according to the proportion of the number of examples in the current batch w.r.t. the number of examples encountered so far. A batch specific value λ_i can be obtained as:

$$\lambda_i = \begin{cases} 0, & i = 1 \\ map(1 - \frac{n_i}{\sum_{j \leq i} n_j}), & i > 1 \end{cases} \tag{5}$$

where map is a linear mapping function that can shift and stretch/compress its input. For example, considering the number of classes in CORe50 (i.e. 10 in the first batch and 5 in the successive batches) and assuming that map is the identity function, we obtain: $\lambda_1 = 0, \lambda_2 = \frac{2}{3}, \lambda_1 = \frac{3}{4}, \ldots, \lambda_9 = \frac{9}{10}$.

Another important facet is the learning strength to adopt in the initial batch B_1 and successive batches B_i. It is worth noting that in LWF (as for EWC and SI) training on incremental batches $B_i, i > 1$ should not be forced to convergence. In fact, as the regularization part of the loss becomes dominant the training accuracy tend to decrease and trying to leverage it with aggressive learning rates and high number of epochs can lead to divergence. In our experiments, we trained the model on each batch for a fixed small number of epochs without forcing convergence. Using a simple early stopping criteria is crucial for continuous learning because of efficiency and lack of realistic validation sets.

Summarizing, LWF implementation with weighted soft target vectors is very simple and, for each batch $B_i, i > 1$, its overhead consists of `i)` computation: one extra forward pass for each of the n_i pattern; `ii)` storage: temporary storing (for the batch lifespan) the $\hat{y}_{lwf}$ predictions, consisting of $n_i \cdot s$ values.

2.2 Elastic Weight Consolidation (EWC)

EWC [14] is a regularization approach which tries to control forgetting by selectively constraining (i.e., freezing to some extent) the model weights which are important for the previous tasks.

Intuitively, once a model has been trained on a task, thus reaching a minimum in the loss surface, the sensitivity of the model w.r.t. each of its weight θ_k can be estimated by looking at the curvature of the loss surface along the direction determined by θ_k changes. In fact, high curvature means that a slight θ_k change results in a sharp increase of the loss. The diagonal of the Fisher information matrix F, which can be computed from first-order derivatives alone, is equivalent to the second derivative (i.e. curvature) of the loss near a minimum. Therefore, the k^{th} diagonal element in F (hereafter denoted as F_k) denotes the importance of weight θ_k. Important weights must be moved as little as possible when the model is fine-tuned on new tasks. In a two tasks scenario this can be achieved by adding a regularization term to the loss function when training on the second task:

$$L = L_{cross}(\hat{y}, t = \hat{y}_{1h}) + \frac{\lambda}{2} \cdot \sum_k F_k(\theta_k - \theta_k^*)^2 \tag{6}$$

where:

[3] In general the suitability of a weighting scheme, can be assessed by looking at the evolution of confusion matrices (see Figure 11 in Section 5). Sophisticated weighted schemes tend to overfit the particular batch sequences and must be adopted with care. Eq. 5 is quite simple and has no hyperparameters to tune.

- θ_k^* are the optimal weight values resulting from the first task.
- λ is the regularization strength.

Let us now consider a sequence of tasks or batches B_i. After training the model on batch B_i we need to compute the Fisher information matrix F^i and store the set of optimal weights Θ^i. F_i and Θ^i will be then used to regularize the training on B_{i+1}. Each diagonal element F_k^i can be computed as the variance of $\partial L_{cross}(\hat{y}, t)/\partial\theta_k$ over the n_i patterns of B_i.

Two different EWC implementations can be setup in practice:

1. A distinct regulation term is added to the loss function for each old task. This requires maintaining a Fisher matrix F^i and a set of optimal weights Θ^i for each of the previous tasks/batches;
2. A single Fisher matrix F is initialized to 0 and consolidated at the end of a batch B_i by (element wise) summing the Fisher information: $F = F + F^i$. A single set of optimal weights Θ is also maintained by using the most recent ones ($\Theta = \Theta^i$) since Θ^i already incorporates constraints from all previous batches (refer to the discussion in [9][15]).

Option 1 can be advantageous to precisely control EWC training dynamic in the MT scenario with few tasks, but is not practical (because of storage and computation issues) in SIT scenario with several batches. It is worth noting that in option 2, the F_k values can only increase as new batches are processed, potentially leading to divergence for large λ. To better understand this issue, let us consider how the regularization term is dealt with by gradient descent: this is quite similar to L2 regularization and can be implemented as a special weight decay where weights θ_k are not decayed toward 0, but toward θ_k^*. The weight update determined by the loss function (eq. 6) is:

$$\theta_k' = \theta_k - \eta \cdot \frac{\partial L_{cross}(\hat{y}, t)}{\partial \theta_k} - \eta \cdot F_k(\theta_k - \theta_k^*) \tag{7}$$

where η is the learning rate. In the above equation if, for some k, the product $\eta \cdot \lambda \cdot F_k$ is greater than 1, the weight correction toward θ_k^* is excessive and we overshoot the desired value. Tuning λ according to the maximum theoretical value of F_k is problematic because: `i)` we do not know such value; `ii)` using a too high value might lead to unsatisfactory performance since does not allow to constrain the weights associated to mid-range F_k enough. We empirically found that a feasible solution is normalizing F after each batch B_i as:

$$\begin{aligned} F &= F + F^i \\ \hat{F} &= clip(\frac{F}{i}, max_F) \end{aligned} \tag{8}$$

where $clip$ sets the matrix values exceeding max_F to the constant max_F. Note that F/i replaces the Fisher matrix sum with an average, and this could be counterintuitive. Let us suppose that weight θ_5 is very important for batch B_1 and this is reflected by an high value of F_5^1, then if θ_5 is not important for B_2 as well (i.e., F_5^2 is small) computing the average $1/2 \cdot (F_5^1 + F_5^2)$ pulls down the combined importance. However, this can be compensated by a proper selection of a max_F in order to saturate $\hat{F}$ values even for those weights which are important for a single task. Given max_F and η we can easily determine the maximum value for λ as $1/(\eta \cdot max_F)$.

An example is shown in Figure 4, where the distribution of Fisher information values is reported after B_1, B_2 and B_3. In the first row F values denotes a long tail on the right. In the second row, F_k values are averaged and clipped to 0.001 thus allowing to work with higher λ and better control forgetting.

Summarizing, EWC implementation is moderately simple and, for each batch B_i, its overhead consists of `i)` computation of Fisher information F^i, requiring one forward and one backward propagation for each of the n_i patterns; `ii)` storage of F and Θ, totaling $2 \cdot m$ values, where m is the number of model weights (including biases).

2.3 Synaptic Intelligence

SI was introduced in [33] as a variant of EWC. The authors argued that computation of Fisher information is expensive for continuous learning and proposed to calculate weight importance on-line during SGD.

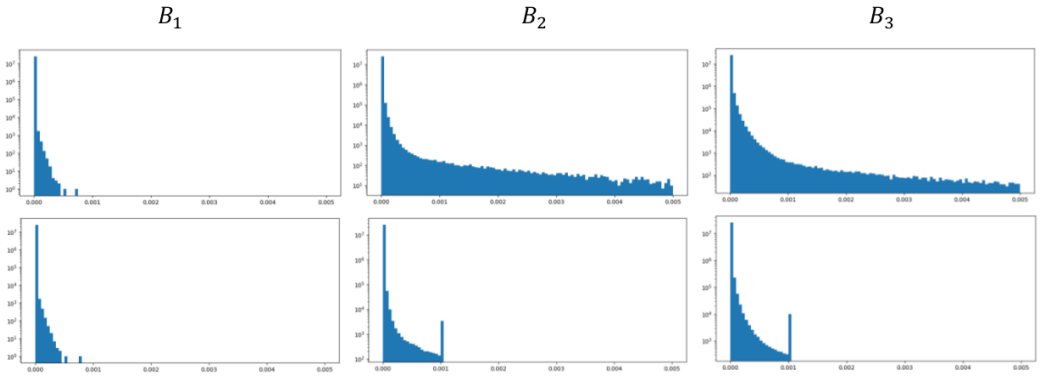

Figure 4: CaffeNet trained by EWC on CORe50 SIT (details in Section 4.2). The first row shows F values distribution denoting a long tail on the right: considering the logarithmic scale, the number of weights values taking high values in F is quite limited. The second row shows the normalized matrix $\hat{F}$ obtained by averaging F values and max clipping to 0.001. Saturation to 0.001 is well evident, but after B_3 the fraction of saturated weights is small (about 1/1000).

The loss change given by a single weight update step during SGD is given by:

$$\Delta L_k = \Delta\theta_k \cdot \frac{\partial L}{\partial \theta_k} \tag{9}$$

where $\Delta\theta_k = \theta'_k - \theta_k$ is the weight update amount and $\partial L/\partial\theta_k$ the gradient. The total loss change associated to a single parameter θ_k can be obtained as running sum $\sum \Delta L_k$ over the weight trajectory (i.e., the sequence of weight update steps during the training on a batch). The weight importance (here denoted as F_k to keep notation uniform with previous section) is then computed as:

$$F_k = \frac{\sum \Delta L_k}{T_k^2 + \xi} \tag{10}$$

where T_k is the total movement of weight θ_k during the training on a batch (i.e., the difference between its final and the initial value) and ξ is a small constant to avoid division by 0 (see [33] for more details). Note that the whole data needed to calculate F_k is available during SGD and no extra computation is needed.

In the SIT scenario we empirically found that an effective normalization after each batch B_i is:

$$\begin{aligned} F &= F + w_i \cdot F^i \\ \hat{F} &= clip(F, max_F) \end{aligned} \tag{11}$$

where F is set to 0 before first batch and then consolidated as a weighted sum with batch specific weights w_i. Actually in our experiments, as reported in Section 5, we used a small value w_1 for the first batch and a constant higher value for all successive batches: $w_2 = w_3 = \cdots = w_9$. Considering CORe50 experiments, since in the first batch we tune a model from ImageNet weight initialization, the trajectories that most of the weights have to cover to adapt to CORe50 are longer than for successive batches whose tuning is intra dataset. This is not the case for EWC, because EWC looks at the loss surface at convergence, independently of the length of weight trajectories.

Given $\hat{F}$ values, SI regularization can be implemented as EWC. The magnitude of $\hat{F}$ values can also be made comparable to EWC by proper setting of w_i, so that max_F and λ can take similar values. Figure 5 compares the distributions of $\hat{F}_k$ values between EWC and SI: at first glance the distributions appear to be similar; of course more precise correlation studies could be performed, but this is out of the scope of this work.

Summarizing, SI implementation is quite simple and, for each batch B_i, its overhead consists of `i)` computation of weight importance F^i, based on information already available during SGD; `ii)` storage of F and Θ, totaling $2 \cdot m$ values, where m is the number of model weights.

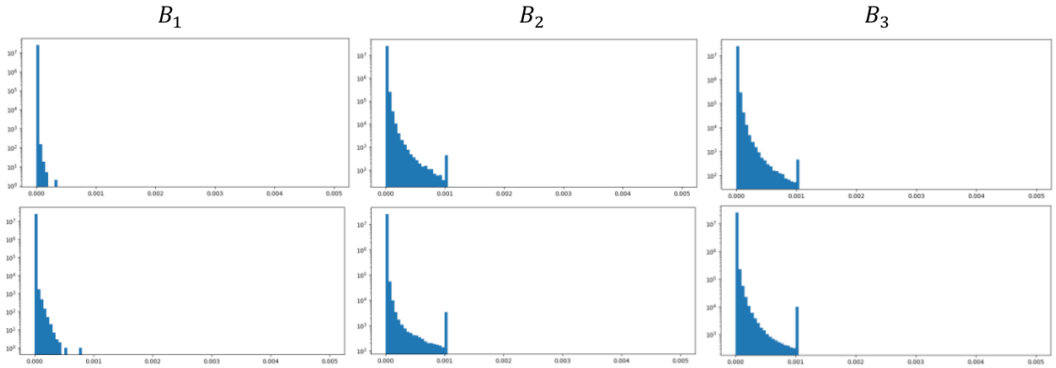

Figure 5: CaffeNet trained on CORe50 SIT. The first row shows $\hat{F}$ values distribution obtained by SI on batches B_1, B_2 and B_3. $\hat{F}$ values distribution from EWC is reported in the second row for comparison. The shape of the distribution is quite similar, even if in this experiments, the number of SI saturated values is about 10 times lower.

3 AR1: Combining Architectural and Regularization Strategies

In this section we introduce AR1, a novel continuous learning approach obtained by combining architectural and regularization strategies. We start by reviewing CWR, a simple architectural technique that we introduced in [19] and that, in spite of its simplicity, proved to be effective in SIT scenario. Then we first improve CWR with two simple but valuable modifications (that is, mean-shift and zero initialization) and finally extend it by combining it with SI.

3.1 Copy Weight with Reinit (CWR)

CWR was proposed in [19] as baseline technique for continuous learning from sequential batches. While it can work both for NC (new classes) and NIC (new instances and classes) update content type, here we focus on NC under SIT scenario.

Referring to Figure 2 (bottom) the most obvious approach to implement a SIT strategy seems to be:

1. Freeze shared weights $\bar{\Theta}$ after the first batch.
2. For each batch B_i, extend the output layers with new neurons/weights for the new classes, randomly initialize the new weights but retain the optimal values for the old class weights. The old weights could then be frozen (denoted as FW in [20]) or continued to be tuned (denoted as CW in [20]).

Implementing step 2 as above proved to be suboptimal with respect to CWR approach (see Figure 8 of [20] for a comparison) where old class weights are re-initialized at each batch.

Algorithm 1 CWR

$cw = 0$
init $\bar{\Theta}$ random or from a pre-trained model (e.g. ImageNet)
for each training batch B_i:
    expand output layer with s_i neurons for the new classes in B_i
    random re-init tw (for all neurons in the output layer)
    Train the model with SGD on the s_i classes of B_i:
        if $B_i = B_1$ learn both $\bar{\Theta}$ and tw
        else learn tw while keeping $\bar{\Theta}$ fixed
    for each class j among the s_i classes in B_i:
        $cw[j] = w_i \cdot tw[j]$
    Test the model by using $\bar{\Theta}$ and cw

To learn class-specific weights without interference among batches, CWR maintains two sets of weights for the output classification layer: cw are the consolidated weights used for inference and tw

the temporary weights used for training: cw are initialized to 0 before the first batch, while tw are randomly re-initialized (e.g., Gaussian initialization with std = 0.01, mean = 0) before each training batch. At the end of each batch training, the weights in tw corresponding to the classes in the current batch are scaled and copied in cw: this is trivial in NC case because of the class segregation in different batches but is also possible for more complex cases (see Section 5.3 of [19]). To avoid forgetting in the lower levels, after the first batch B_1, all the lower level weights $\bar{\Theta}$ are frozen. Weight scaling (with batch specific weights w_i) is necessary in case of unbalanced batches with respect to the number of classes or number of examples per class.

More formally, let $cw[j]$ and $tw[j]$ be the subset[4] of weights related to class j, then CWR learning sequence can be implemented as described in Algorithm 1.

In CWR experiments reported in [19], to better disentangle class-specific weights we used models without class-shared fully connected layers (e.g., we removed FC6 and FC7 in CaffeNet). In fact, since $\bar{\Theta}$ weights are frozen after the first batch, fully connected layer weights tend to specialize on the first batch classes only. However, since skipping fully connected layers is not mandatory for CWR, in this paper to better compare different approaches we prefer to focus on native models and keep fully connected layers whether they exist. Finally, CWR implementation is very simple and, the extra computation is negligible and for each batch B_i, its overhead consists of the storage of temporary weights tw, totaling $s \cdot pn$ values, where s is the number of classes and pn the number of penultimate layer neurons.

3.2 Mean-shift and Zero Initialization (CWR+)

Here we propose two simple modifications of CWR: the resulting approach is denoted as CWR+. The first modification, mean-shift, is an automatic compensation of batch weights w_i. In fact, tuning such parameters is annoying and a wrong parametrization can lead the model to underperform. We empirically found that, if the weights tw learnt during batch B_i, are normalized by subtracting their global average, then rescaling by w_i is no longer necessary (i.e., all $w_i = 1$). Other reasonable forms or normalization, such as setting standard deviation to 1, led to worse results in our experiments.

Algorithm 2 CWR+

$cw = 0$
init $\bar{\Theta}$ random or from a pre-trained model (e.g. ImageNet)
for each training batch B_i:
    expand output layer with s_i neurons for the new classes in B_i
    $\mathbf{tw} = \mathbf{0}$ (for all neurons in the output layer)
    Train the model with SGD on the s_i classes of B_i:
        if $B_i = B_1$ learn both $\bar{\Theta}$ and tw
        else learn tw while keeping $\bar{\Theta}$ fixed
    for each class j among the s_i classes in B_i:
        $\mathbf{cw[j]} = \mathbf{tw[j]} - \mathbf{avg(tw)}$
    Test the model by using $\bar{\Theta}$ and cw

The second modification, denoted as zero init, consists in setting initial weights tw to 0 instead of typical Gaussian or Xavier random initialization. It is well known that neural network weights cannot be initialized to 0, because this would cause intermediate neuron activations to be 0, thus nullifying back-propagation effects. While this is certainly true for intermediate level weights, it is not the case for the output level (see Appendix E for a simple derivation). Actually, what is important here is not using the value 0, but the same value for all the weights: 0 is used for simplicity. Even if this could appear a minor detail, we discovered that it has a significant impact on the training dynamic and the forgetting. If output level weights are initialized with Gaussian or Xavier random initialization they typically take small values around zero, but even with small values in the first training iterations the softmax normalization could produce strong predictions for wrong classes. This would trigger unnecessary errors backpropagation changing weights more than necessary. While this initial adjustment is uninfluential for normal batch training we empirically found that is detrimental for continuous learning. Even if in this paper we apply zero init only to CWR+ and AR1, we found

[4] the number of weights in each subset typically corresponds to the number of neurons in the penultimate layer.

that even a simple approach such as fine tuning can greatly benefit from zero init in case of continuous learning.

In Algorithm 2 we report the pseudocode for CWR+: the modifications w.r.t. CWR are highlighted in bold. CWR+ overhead is basically the same of CWR since taking the average of tw is computationally negligible w.r.t. the SGD complexity.

3.3 AR1

A drawback of CWR and CWR+ is that weights $\bar{\Theta}$ are tuned during the first batch and then frozen. AR1, is the combination of an Architectural and Regularization approach. In particular, we extend CWR+ by allowing $\bar{\Theta}$ to be tuned across batches subject to a regularization constraint (as per LWF, ECW or SI). We did several combination experiments on CORe50 to select a regularization approach; each approach required a new hyperparameter tuning w.r.t. the case when it was used in isolation. At the end, our choice for AR1 was in favor of SI because of the following reasons:

- LWF performs nicely in isolation, but in our experiments it does not bring relevant contributions to CWR+. We guess that being the LWF regularization driven by an output stability criterion, most of the regularization effects go to the output level that CWR+ manages apart.
- Both EWC and SI provide positive contributions to CWR+ and their difference is minor. While SI can be sometime unstable when operating in isolation (see Sections 4.2 and 4.3) we found it much more stable and easy to tune when combined with CWR+.
- SI overhead is small, since the computation of trajectories can be easily implemented from data already computed by SGD.

In Algorithm 3 we report the pseudocode for AR1.

Algorithm 3 AR1

$cw = 0$
init $\bar{\Theta}$ random or from a pre-trained model (e.g. ImageNet)
$\Theta = 0$ (Θ are the optimal shared weights resulting from the last training, see Section 2.3)
$\hat{F} = 0$ ($\hat{F}$ is the weight importance matrix, see Section 2.3).
for each training batch B_i:
    expand output layer with s_i neurons for the new classes in B_i
    $tw = 0$ (for all neurons in the output layer)
    Train the model with SGD on the s_i classes of B_i by simultaneously:
        learn tw with no regularization
        learn $\bar{\Theta}$ subject to SI regularization according to $\hat{F}$ and Θ
    for each class j among the s_i classes in B_i:
        $cw[j] = tw[j] - avg(tw)$
    $\Theta = \bar{\Theta}$
    Update $\hat{F}$ according to trajectories computed on B_i (see eq. 10 and 11)
    Test the model by using $\bar{\theta}$ and cw

AR1 overhead is the sum of CWR+ and SI overhead:

- storage:
  - Temporary weights tw, totaling $s \cdot pn$ values, where s is the number of classes and pn the number of penultimate layer neurons.
  - F and Θ, totaling $2 \cdot (m - s \cdot pn)$, where m is the total number of model weights.
- computation:
  - Weights importance $\hat{F}$, based on information already available during SGD.
  - Learning sw subject to SI regularization can be easily implemented as weight decay (see eq. 7) and is computationally light.

Considering the low computational overhead and the fact that typically SGD is typically early stopped after 2 epochs, AR1 is suitable for online implementations.

4 Experiments

4.1 CNN Models for CORe50

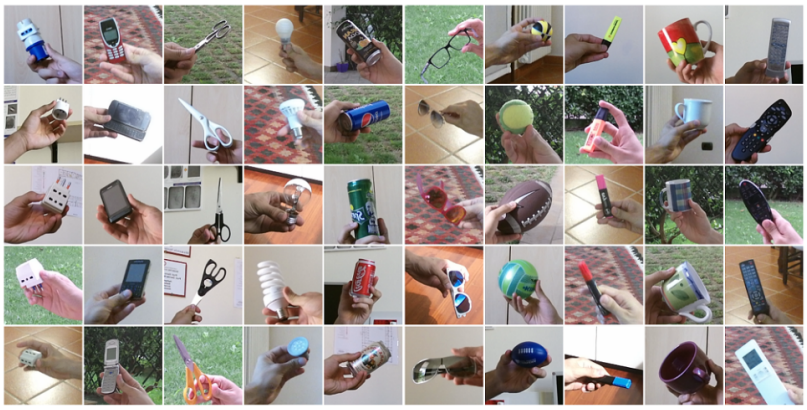

Figure 6: Example images of the 50 objects in CORe50. Each column denotes one of the 10 categories, but for the experiments reported in this paper we associate each object to a distinct class.

The experiments reported in this paper have been carried out with two CNN architectures: CaffeNet [10] and GoogLeNet [31]. Minor modifications have been done to the default models as here detailed:

- CaffeNet and GoogLeNet original models work on input images of size 227×227 and 224×224 respectively. CORe50 images are 128×128 and stretching them to 227×227 or 224×224 is something that should be avoided (would increase storage and amount of computation). On the other hand, as discussed in appendix A of [20], simply reshaping the network input size leads to a relevant accuracy drop. This is mainly due to the reduced resolution of feature maps whose size along the hierarchy is about half the original. We noted that the accuracy can be restored by halving the stride in the first convolutional layer and by adjusting padding if necessary: unfortunately, this also restore much of the original computational complexity, but `i)` does not require unnecessary image stretching, `ii)` allows to save memory, and `iii)` reduces CPU→GPU data transfer when loading mini-batches.
- For CaffeNet, the number of neurons in fc6 and fc7 fully connected layers was halved. This substantially reduce the number of parameters without any relevant accuracy drop.
- GoogLeNet has three output layers. The deepest one is typically used for prediction, while the two intermediate ones are useful to boost gradient flow during training thank to the injection of a fresh supervised signals at intermediate depth. While the deepest output level is proceeded by a global average pooling, each intermediate output layer is preceded by a fully connected layer; in our experiment where GoogLeNet was always initialized from ImageNet such fully connected layers did not provide any advantage, hence to simplify the architecture and reduce the number of parameters we removed them. Finally, note that when GoogLeNet is used with CWR, CWR+, and AR1 we need to maintain in cw a copy of the weights of all the three output levels.

Table 3 in Appendix B lists all the changes between original and modified models. Model weights (in convolutional layers) have been always initialized from ImageNet pre-training. We also tried other popular CNN architectures on CORe50, including NiN [18] and ResNet-50 [6]. Table 2 compares the accuracy of these models when trained on the whole training set (i.e., all training batches combined). The gap between GoogLeNet and ResNet50 is quite small, but the latter is much slower to train, so we decided to use GoogLeNet as a near state-of-the-art model.

4.2 Results on CORe50

CORe50 dataset [19] was specifically designed as a benchmark for continuous object recognition (Figure 6). Here we consider NC content update type (a.k.a. incremental-class learning) where the 50 classes are partitioned in 9 batches provided sequentially: B_1 includes 10 classes while $B_2, \ldots, B_9$

Table 2: Model training on CORe50 by using the whole training set (fusion of all training batches). Models are adapted to work with 128x128 inputs and weights in convolutional layers are initialized from ImageNet pre-training. Time refers to a single Titan X Pascal GPU and Caffe framework.

Model (128x128)	Test accuracy	Mini-batch size	# Epochs	Time (m)
CaffeNet	74.1%	256	4	7
NiN	82.3%	256	4	14
GoogLeNet	91.3%	64	4	30
ResNet-50	92.9%	12	4	120

5 classes each. For each class 2400 patterns (300 frames $\times$ 9 sessions) are included in the training batches and 900 patterns (300 frames $\times$ 3 sessions) are segregated in the test set. The test set is fixed and the accuracy is evaluated after each batch on the whole test set, including still unseen classes. Refer to Section 5.2 of [19] for a discussion about CORe50 testing protocol. Experiments in Section 4.3 confirm the usefulness of reporting results on a fixed test set.

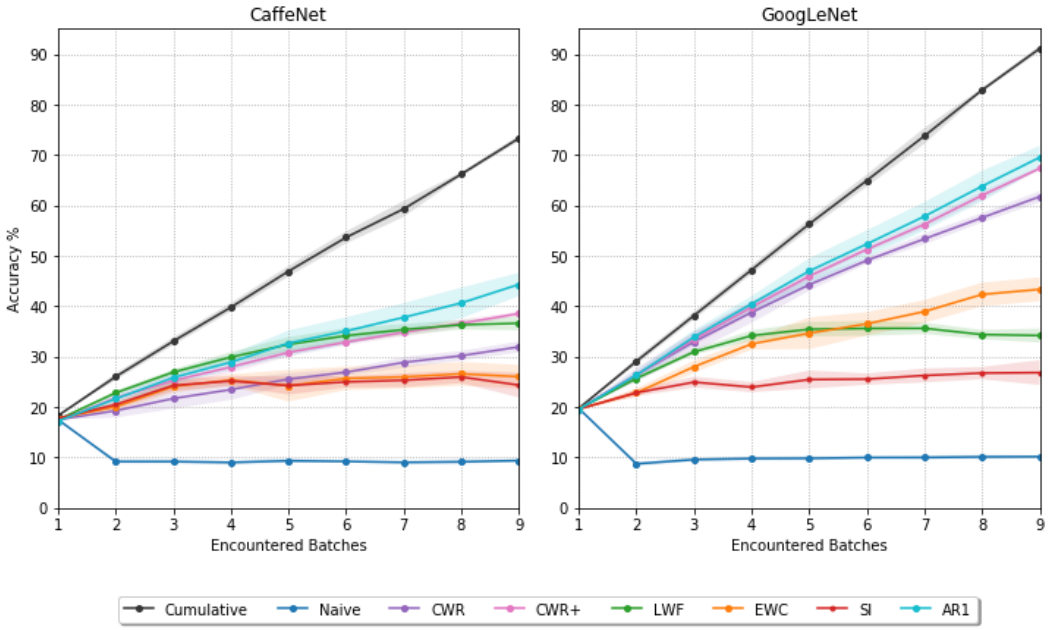

Figure 7: The graphs show CaffeNet and GoogLeNet accuracy on CORe50 after each training batch (average on 10 runs, with different class ordering). Colored areas represent the standard deviation of each curve. Better viewed in color.

Continuous learning approaches introduced in Section 2 and 3 are here compared with two baseline approaches:

1. **Naive**: simply tunes the model across the batches without any specific mechanism to control forgetting, except early stopping.
2. **Cumulative**: the model is retrained with the patterns from the current batch and all the previous batches (only for this approach we assume that all previous data can be stored and reused). This is a sort of upper bound, or ideal performance that continuous learning approaches should try to reach.

LWF, EWC and SI were run with the modifications (variable lambda, normalization, clipping, etc.) introduced in Section 2. In fact, when the approaches were tested in their native form, either we were not able to make them consistently learn across the batches or to contrast forgetting enough.

Figure 7 shows CaffeNet and GoogLeNet accuracy in CORe50, SIT - NC scenario. Accuracy is the average over 10 runs with different batch ordering. For all the approaches hyperparameter tuning was

performed on run 1, and then hyperparameters were fixed across runs $2, \ldots, 10$. Table 4 in Appendix C shows hyperparameter values for all the methods. From these results we can observe that:

- The effect of catastrophic forgetting is well evident for the **Naïve** approach: accuracy start from 17-20% at the end of B_1 (in line with the 20% of classes in B_1), but then suddenly drops to about 9-10% (that is, the proportion of classes in each subsequent batch); in other words, as expected, after each batch, the model tends to completely forget previous classes while learning the new ones.
- **LWF** behaves well for CaffeNet and moderately well for GoogLeNet: continuous learning is evident and accuracy is much better than Naive approach. A few percentage point drop can be noted in the last batches for GoogLeNet; to avoid this we tried to boost stability by increasing the lambda value: this yielded to an increasing learning trend across all the batches but the top accuracy was never higher than 32% (neither in the central batches nor in the last ones), so we decided to rollback to previous parametrization.
- Both the models are able to learn incrementally with **EWC**, but while for GoogLeNet the learning trend is quite good, CaffeNet after a few batches tends to stabilize and the final accuracy is much lower. Since GoogLeNet is a deeper and more powerful network we can expect a certain accuracy gap (see the corresponding cumulative approach curves); however here the gap is much higher than for LWF and in our experiment we noted that EWC (and SI) tend to suffer more than LWF the presence of fully connected layers such as FC6 and FC7 layer in CaffeNet[5]. Fully connected layers are usually close to the final classification layer, and any change in their weights is likely to have a major impact on classification. Even if EWC can operate on all layers, precisely constraining the weights of fully connected layers appears to be challenging due to their high importance for all tasks. LWF, whose regularization signal "comes from the top", seems to better deal with fully connected layers. To further investigate this issue some experiments have been performed by removing FC6 and FC7 from CaffeNet and, in spite of the shallower and less powerful network, for EWC we got a few percentage improvements while LWF accuracy slightly decreased.
- While CaffeNet accuracy with **SI** is very close to EWC, GoogLeNet accuracy with SI is markedly lower than EWC. In general, we noted that SI is less stable than EWC and we believe that SI weights importance estimation can be sometimes less accurate than the EWC one because of the following reasons:
  - A weight which is not moved by SGD is not considered important for a task by SI, but this is not always true. Let us assume that Batch B_1 trains the model on classes $c_1, c_2, \ldots, c_{10}$; the output layer weights which are not connected to the above class output neurons, are probably left unchanged to their (near 0) initial value; however, this does not mean that they are not important for B_1 because if we raise their values the classification might dramatically change. More in general, if a weight already has the right value for a task and is not moved by SGD, concluding that it is not important for the task can be sometime incorrect.
  - A weight could cover a long trajectory and, at the end of the training on a batch, assume a value similar to the initial one (closed trajectory). Such situation can happen because the loss surface is highly non convex and gradient descent could increase a weight while entering a local minimum and successively restoring its value once escaped.
- **CWR** and **CWR+** accuracy is quite good, always better than LWF, EWC, SI on GoogLeNet, and better than EWC and SI on CaffeNet. Continuous learning trend is here nearly linear across batches, with no evident saturation effect. The relevant improvement of CWR+ over CWR is mainly due to zero init.
- **AR1** exhibits the best performance. SI regularization is here quite stable and pushes up CWR+ accuracy. AR1 is also stable w.r.t. parameter tuning: in Table 4 one can observe that we used almost the same hyperparameters for CaffeNet and GoogLeNet. For GoogleNet AR1 reaches a remarkable accuracy of about 70% with small standard deviation among runs, and the gap w.r.t. the Cumulative approach is not large, thus proving that continuous learning (without storing/reusing old patterns) is feasible in a complex incremental class scenario.

[5]GoogleNet, as many modern network architectures, does not include fully connected layers before the last classification layer.

4.3 Results on iCIFAR-100

CIFAR-100 [16] is a well-known and largely used dataset for small (32×32) natural image classification. It includes 100 classes containing 600 images each (500 training + 100 test). The default classification benchmark can be translated into a SIT-NC scenario (denoted as iCIFAR-100 by [28]) by splitting the 100 classes in groups. In this paper we consider groups of 10 classes thus obtaining 10 incremental batches.

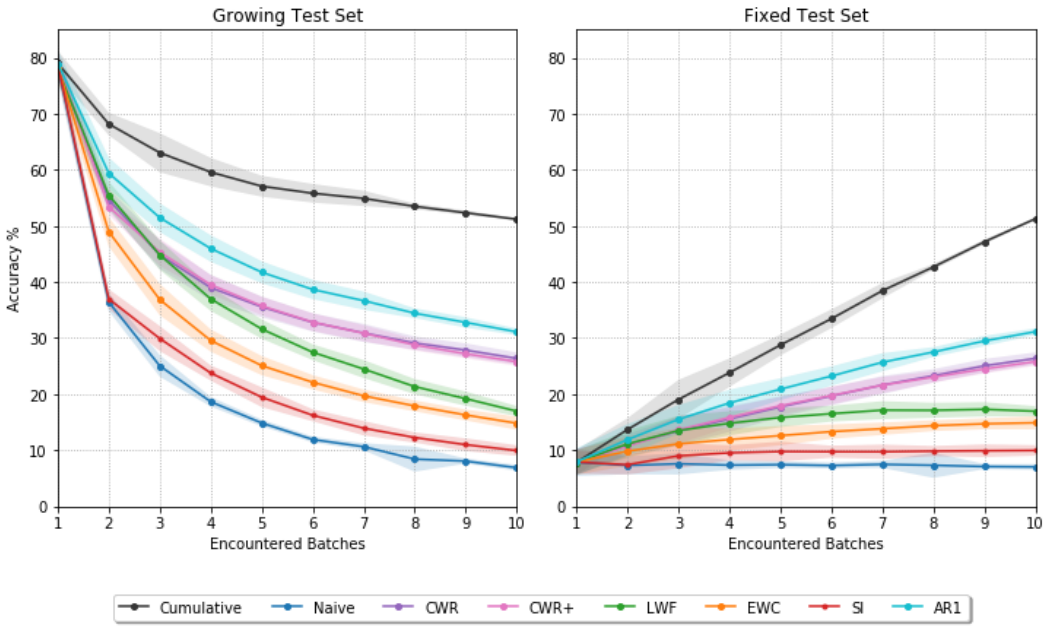

Figure 8: Accuracy on iCIFAR-100 with 10 batches (10 classes per batch). Results are averaged on 10 runs: for all the strategies hyperparameters have been tuned on run 1 and kept fixed in the other runs. The experiment on the right, consistently with CORe50 test protocol, considers a fixed test set including all the 100 classes, while on the left we include in the test set only the classes encountered so far (analogously to results reported in [28]). Colored areas represent the standard deviation of each curve. Better viewed in color.

The CNN model used for this experiment is the same used by [33] for experiments on CIFAR-10/100 Split and whose results have been reported in Figure 3 for the MT scenario. It consist of 4 convolutional + 2 fully connected layers; details are available in Appendix A of [33]. The model was pre-trained on CIFAR-10 [16]. Figure 8 compares the accuracy of the different approaches on iCIFAR-100. The results obtained on CORe50 are almost confirmed, in particular:

- Unlike the **Naïve** approach, **LWF** and **EWC** provide some robustness against forgetting, even if in this incremental scenario their performance is not satisfactory. **SI**, when used in isolation, is quite unstable and performs worse than LWF and EWC.
- The accuracy improvement of **CWR+** over **CWR** is here very small, because the batches are balanced (so weight normalization is not required) and the CNN initialization for the last level weights was already very close to 0 (we used the authors' default setting of a Gaussian with std = 0.005).
- **AR1** consistently outperforms all the other approaches.

It is worth noting that both the experiments reported in Figure 8 (i.e., fixed and expanding test set) lead to the same conclusions in terms of relative ranking among approaches, however we believe that a fixed test set allows to better appreciate the incremental learning trend and its peculiarities (saturation, forgetting, etc.) because the classification complexity (which is proportional to the number of classes) remains constant across the batches. For example, in the right graph it can be noted that SI, EWC and LWF learning capacities tend to saturate after 6-7 batches while CWR, CWR+ and AR1 continue to grow; the same information is not evident on the left because of the underlying negative trend due to the increasing problem complexity.

Finally note that absolute accuracy on iCIFAR-100 cannot be directly compared with [28] because the CNN model used in [28] is a ResNet-32, which is much more accurate than the model here used: on the full training set the model here used achieves about 51% accuracy while ResNet-32 about 68,1% [24].

4.4 Preliminary Comparison with Rehearsal-based Approaches

While the focus of the paper, as pointed out in Section 2, is on architectural/regularization strategies (i.e., strategies not storing any past input data), in this section we present a preliminary comparison of AR1 with three representative rehearsal-based approaches: basic rehearsal[6] (REHE), GEM [21] and ICARL [28]. Results contained in this section represent a preliminary assessment, since we did not re-implement GEM and ICARL as the other regularization techniques but adapted the implementations made available by their authors to the CORe50 benchmark and the SIT-NC scenario leaving most of the hyperparameters and implementation choices unchanged. In particular, GEM was specifically designed for a Multi-Task setting, and, to the best of our knowledge these are the first experiments on more complex benchmarks such as CORe50 and a Single-Incremental-Task scenario with images larger than 32x32.

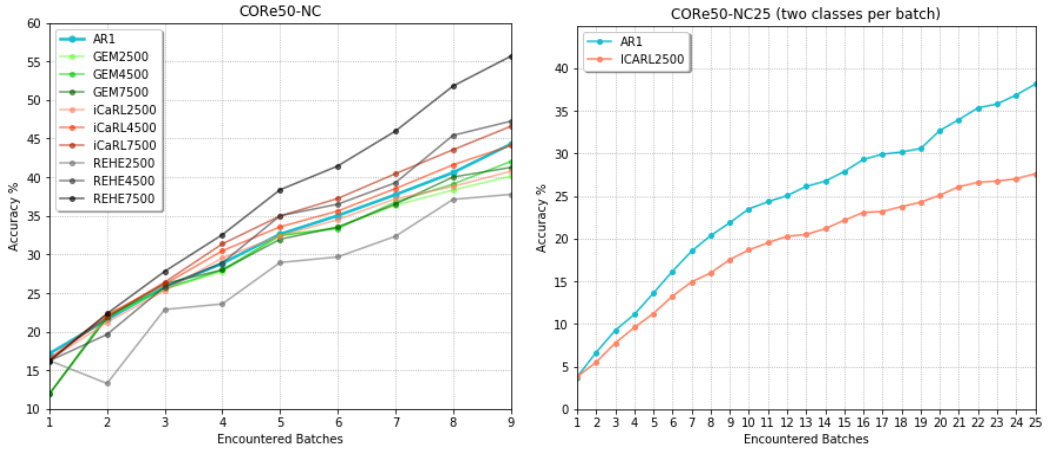

Figure 9: On the left, a comparison of AR1 with rehearsal strategies with different amount of external memory (2500, 4500 and 7500 input patterns). Accuracy is averaged over 3 runs. On the right, a comparison of AR1 with ICARL on a more complex setting where the original 50 classes are split in 25 batches with 2 classes each. For this experiment we also computed the backward transfer metric [21] which is -15,64% for AR1 and -11,09% for ICARL.

In Figure 9 (left), we report the average accuracy of 3 rehearsal-based strategies: REHE, GEM and ICARL each with external memory size of 2500, 4500 and 7500 input patterns (corresponding to about 2%, 4% and 6% of the total training images). We note that AR1, while not storing any training example from the past, is competitive with respect the other approaches, especially when a high memory and computational budget is not available. It is worth noting that the external memory is used differently by ICARL and GEM; in fact, while ICARL is designed to fill the total storage after every batch, GEM uses a fixed amount of memory for each batch and reaches the total memory budget only at the end of the last batch.

In Figure 9 (right), we compare AR1 an ICARL on a more difficult setting with respect to the original CORe50-NC benchmark; the 50 classes are here split in 25 batches with 2 classes each. It is interesting to observe that even if the more challenging problem leads to a certain degradation in terms of absolute accuracy, both techniques perform reasonably well also with smaller batches.

[6]The external memory is filled by sampling past examples from both the external memory and the current training batch. The sampling probability is tuned to balance the number of samples drawn from different batches.

5 Practical Advices for Hyperparameters Tuning

Hyperameters tuning is not easy for complex deep architectures and is still more complex for continuous learning over sequential batches. Simply looking at the accuracy trend along the batches is not enough to understand if an approach is properly parametrized. For example, a poor accuracy can be due to insufficient learning of the new classes or by the forgetting of the old ones.

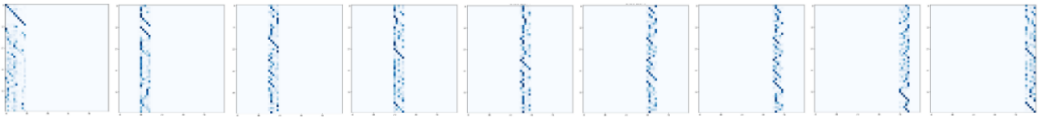

Figure 10: Sequence of confusion matrices computed after each training batch for the Naïve approach on CaffeNet. On the vertical axis the true class and on the abscissa the predicted class.

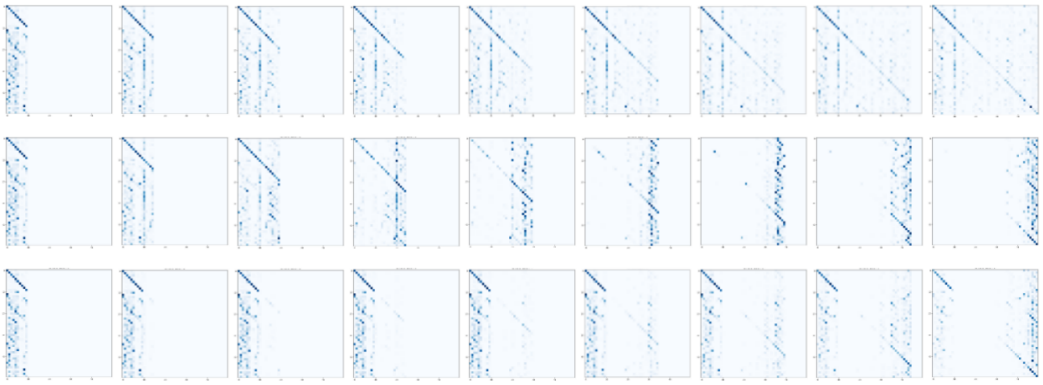

Figure 11: Sequence of confusion matrices computed after each training batch $(1, \ldots, 9)$ for the LWF approach and CaffeNet. In the first row the approach is properly parametrized (variable λ and map function, see Table 4) and the model is able to continuously learn new classes without forgetting the old ones. In the second we used the same $\lambda_i = 0.5$ for all the batches: for $B_2, \ldots, B_3$ the regularization is appropriate but for successive batches is too light leading to excessive forgetting. In the third row we used the same $\lambda_i = 0.8$ for all the batches: for $B_2, \ldots, B_6$ the regularization is too strong and learning of corresponding classes is poor.

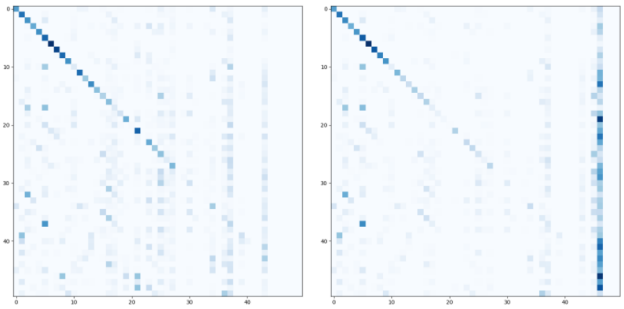

Figure 12: Confusion matrices after computed after B_8 and B_9 for the EWC approach and CaffeNet. As discussed in Section 4.2 EWC tends to saturate CaffeNet capacity after the first 5-6 batches. In this specific run the amount of training on the last batch is too high w.r.t. the residual model capacity and the learning result are poor: the sharp vertical band on the right is an alarm signal.

In our experience visualizing the confusion matrices (CM) is very important to understand what is happening behind the scenes. Looking at the last CM (that is after the last batch) is often not sufficient and the entire CM sequence must be considered. Figure 10 shows the CM sequence (one after each batch) for the naïve approach: forgetting is clearly highlighted by a vertical band moving from the left to the right to cover the classes of the most recent batch. Figure 11 shows three CM sequences

for LWF approach on CaffeNet: `i)` in the first row parametrization is good; `ii)` in the second row the model forget to much old classes, regularization should be increases; `iii)` in the third row initial regularization is too strong and the model cannot learn classes in the corresponding batches.

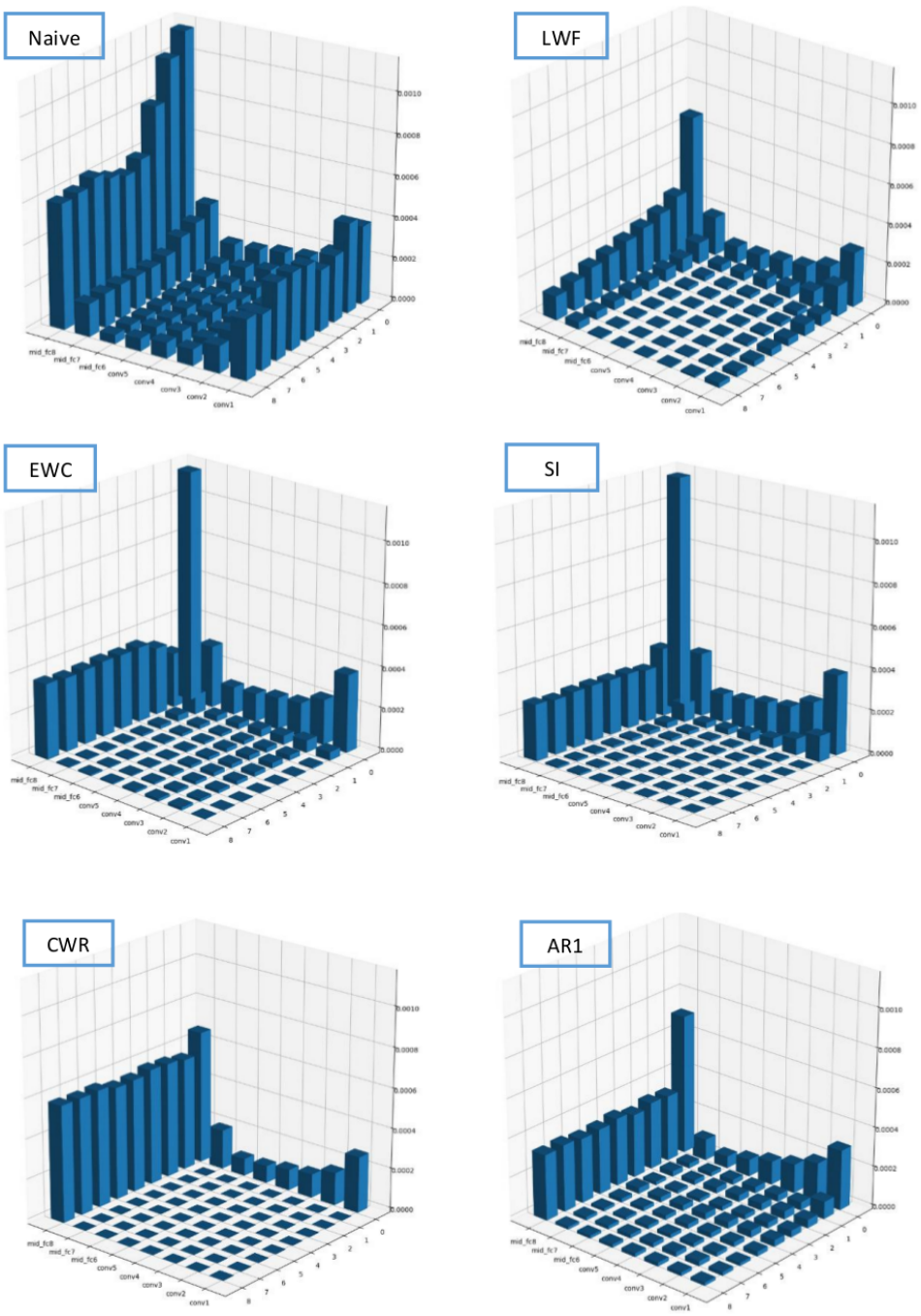

Figure 13: Amount of weight changes by layer and training batch in CaffeNet for different approaches.

Unfortunately, the trade-off stability/plasticity does not depends only on the regularization strength (e.g. λ_i for LWF) because the learning rate and the number of training epochs are also indirectly related. However, looking at the CM sequence allows to understand what is the direction of change of one or more parameters. It can also happen that the amount of regularization is good for the first batches, but unsatisfactory for the successive ones; the CM sequence easily reveals this and allows to

take countermeasures (e.g. the map function for LWF). Finally, when a model is strongly regularized its learning capacity tends to saturate (e.g. in EWC there are no "free" parameters to move in order to learn new classes); this is usually reflected by anomalies such as sharp vertical bars in correspondence of one or few classes (see Figure 12).

Another useful diagnostic technique is visualizing (in the form of a 3D histogram) the average amount of change of the weights in each layer at the end of each batch. To avoid cancellation due to different sign, we compute the average of absolute values. Figure 13 shows the histograms for CaffeNet. We can observe that in the naïve approach weights (which are not constrained) are significantly changed throughout all the layers for all batches. On the other hand, in the regularization approaches, changes tend to progressively reduce batch by batch, and most of the chances occur in the top layers. While in CWR the $\bar{\Theta}$ freeze is well evident, in AR1 intermediate layers weights are moved (more than in EWC and SI) without negative impact in terms of forgetting.

6 Conclusions

In this paper we introduced a novel approach (AR1) for SIT scenario combining architectural and regularization strategies. Results on CORe50 and iCIFAR-100 proves that AR1 allows to train sequentially complex models such as CaffeNet and GoogLeNet by limiting the detrimental effects of catastrophic forgetting.

AR1 accuracy was higher than existing regularization approaches such as LWF, EWC and SI and, from preliminary experiments, compares favorably with rehearsal techniques when the external memory size is not large. AR1 overhead in terms of storage is very limited and most of the extra computation is based on information made available by stochastic gradient descent. We showed that early stopping SGD after very few epochs (e.g., 2) is sufficient to incrementally learn new data on CORe50. Further ideas could be investigated in the future to quantify weight importance for old tasks such as exploiting the moving average of squared gradients already considered by methods like RMSprop [7] or Adam [13] or the Hebbian-like reinforcements between active neurons recently proposed in [1].

In AR1, changing the underlying representation while the model learns new classes introduces some forgetting for the old ones; our experiments confirm that limiting the amount of change of important weights for the old classes, is quite effective to control forgetting. However, if we continue to add classes the network capacity tends to saturate and the representation cannot be further adapted to new data. To deal with this problem we believe that the incremental lateral expansion of some intermediate layers, similar to that proposed in [32] can be a valid approach. Class-incremental learning (NC update content type) is only one of the cases of interest in SIT. New instances and classes (NIC) update content type, available under CORe50, is a more realistic scenario for real applications, and would constitute the main target of our future research.

AR1 extension to unsupervised (or semi-supervised) implementations, such as those described in [22] and [26] is another scenario of interest for future studies. In particular, the 2-level self-organizing model (SOM) proposed by [26] implements memory replay in a very effective way and is capable of exploiting temporal coherence in CORe50 videos with weak supervision. We believe that such a memory replay mechanism could be very effective in conjunction with AR1 and we are going to investigate if a similar mechanism could be embedded in a CNN without external SOM components.

In conclusion, although much more validations in complex setting and new better approaches will be necessary, based on these preliminary results we can optimistically envision a new generation of systems and applications that once deployed continue to acquire new skills and knowledge, without the need of being retrained from scratch.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory Aware Synapses: Learning what (not) to forget. *ArXiv preprint arXiv:1711.09601v2*, page 16, 2017.

[2] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[3] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks. *arXiv preprint arXiv:1312.6211*, 2013.

[4] Stephen Grossberg. Adaptive Resonance Theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, 37:1–47, 2013.

[5] Tyler L. Hayes, Nathan D. Cahill, and Christopher Kanan. Memory Efficient Experience Replay for Streaming Learning. *arXiv preprint arXiv:1809.05922*, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 4, pages 770–778. IEEE, jun 2016.

[7] Geoffrey Hinton. Lecture 6d: a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning. Technical report, Slides of Lecture Neural Networks for Machine Learning, 2012.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *NIPS Deep Learning and Representation Learning Workshop (2015)*, pages 1–9, 2015.

[9] Ferenc Huszár. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences of the United States of America*, 115(11):E2496–E2497, mar 2018.

[10] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *Proceedings of the ACM International Conference on Multimedia*, pages 675–678, 2014.

[11] Ronald Kemker and Christopher Kanan. FearNet: Brain-Inspired Model For Incremental Learning. In *International Conference on Learning Representations (ICLR2018)*, pages 1–16, Vancouver, Canada, 2018.

[12] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring Catastrophic Forgetting in Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI-18)*, New Orleans, USA, 2018.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, pages 1–15, 2014.

[14] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. 114(13):3521–3526, 2017.

[15] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Reply to Huszár: The elastic weight consolidation penalty is empirically valid. *Proceedings of the National Academy of Sciences of the United States of America*, 115(11):E2498, mar 2018.

[16] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. PhD thesis, 2009.

[17] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *14th European Conference on Computer Vision (ECCV 2016)*, volume 9908 LNCS, pages 614–629, Amsterdam, Netherlands, 2016.

[18] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations (ICLR 2014)*, 2014.

[19] Vincenzo Lomonaco and Davide Maltoni. CORe50: a New Dataset and Benchmark for Continuous Object Recognition. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 17–26. PMLR, 2017.

[20] Vincenzo Lomonaco and Davide Maltoni. CORe50: a New Dataset and Benchmark for Continuous Object Recognition. *arXiv preprint arXiv:1705.03550*, 2017.

[21] David Lopez-paz and Marc'Aurelio Ranzato. Gradient Episodic Memory for Continuum Learning. In *Advances in neural information processing systems (NIPS 2017)*, 2017.

[22] Davide Maltoni and Vincenzo Lomonaco. Semi-supervised Tuning from Temporal Coherence. In *23rd International Conference on Pattern Recognition (ICPR 2016)*, pages 2509–2514, 2016.

[23] Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165, 1989.

[24] Xin Pan. ResNet Tensoflow Reimplementation, 2018.

[25] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual Lifelong Learning with Neural Networks: A Review. *arXiv preprint arXiv:1802.07569*, 2018.

[26] German I. Parisi, Jun Tani, Cornelius Weber, and Stefan Wermter. Lifelong Learning of Spatiotemporal Representations with Dual-Memory Recurrent Self-Organization. *arXiv preprint arXiv:1805.10966*, pages 1–20, 2018.

[27] R Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285–308, apr 1990.

[28] Sylvestre-alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental Classifier and Representation Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, Hawaii, 2017.

[29] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*, 2016.

[30] Peter Sadowski. Notes on Backpropagation, 2016.

[31] C Szegedy, W Liu, Y Jia, and P Sermanet. Going deeper with convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[32] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Growing a Brain : Fine-Tuning by Increasing Model Capacity. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[33] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual Learning Through Synaptic Intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3987–3995, Sydney, Australia, 2017.

A Implementation Details (Caffe framework)

Since implementing dynamic output layer expansion was tricky in Caffe framework, we initially implemented the different strategies by using a single *maximal head* (i.e., including all the problem classes since from the beginning) instead of an *expanding head*. In principle, the two approaches are quite similar, since if a particular batch does not contain patterns of a given class, no relevant error signals are sent back along the corresponding connections during SGD. Hoverer, looking at the details of the training process, the two approaches are not exactly the same.

For example, for CWR+ we verified, with some surprise, that the maximal head simplifying approach constantly leads to better accuracy (up to 6-7% on CORe50) w.r.t. to the expanding head approach. We empirically found that the reason is related to the gradient dynamics during the initial learning iterations: working with a higher number of classes makes initial predictions smaller (because of softmax normalization) and the gradient correction for the true class stronger; in a second stage, predictions start to converge and the gradient magnitude is equivalent in the two approaches. It seems that for SGD learning (with fixed learning rate) boosting the gradient in the first iterations favors accuracy and reduces forgetting. We checked this by experimentally verifying that the expanding head approach combined with a variable learning rate performs similarly to maximal head with fixed learning rate. Therefore, to maximize accuracy and reduce complexity, CWR and its evolutions (CWR+ and AR1) have been implemented with the maximal output layer approach. Referring to the pseudocode in Algorithms 1, 2 and 3, it is sufficient to keep to constant maximum size (e.g., 50 for CORe50) and remove the line "expand output layer with...".

For the other approaches we verified that: LWF performs slightly better with expanding head approach while EWC and SI work better (and are easy to tune) with maximal head. To produce the results

presented in Section 4 we used for each strategy the approach that proved to be the most effective. Strategy specific notes are reported in the following for our Caffe implementation.

LWF. It is worth noting that in Caffe a cross-entropy loss layer accepting soft target vectors is not available in the standard layer catalogue and a custom loss layer need to be created.

EWC. To implement EWC in Caffe we:

- compute, average and clip F^i values in pyCaffe (for maximum flexibility). To calculate F^i_k the variance of the gradient should be computed by taking the gradient of each of the n_i patterns in isolation. To speed-up implementation and improve efficiency we computed the variance at mini-batch level, that is using the average gradients over mini-batches. In our experiment we did not note any performance drop even when using mini-batches of 256 patterns.
- pass F and Θ^* to the solver via a further input layer.
- modified SGD solver, by adding a custom regularization that performs EWC regularization in a weight decay style.

SI. Starting from EWC implementation, SI can be easily setup in Caffe. In fact, the regularization stage is the same and we only need to compute F^i values during SGD. For this purpose, in the current implementation, for maximum flexibility, we used pyCaffe.

B Architectural Changes in the Models Used on CORe50

Table 3: Summary of changes w.r.t. the original CaffeNet and GoogLeNet models used in this paper.

CaffeNet		
Layer	*Original*	*Modified*
data (Input)	size: 227×227	size: 128×128
conv1 (convolutional)	stride: 4	stride: 2
conv2 (convolutional)	pad: 2	pad: 1
fc6 (fully connected)	neurons: 4096	neurons: 2048
fc7 (fully connected)	neurons: 4096	neurons: 2048
fc8 (output)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)
GoogLeNet		
Layer	*Original*	*Modified*
data (Input)	size: 224×224	size: 128×128
conv1/7x7_s2 (convolutional)	stride: 2, pad: 3	stride: 1, pad: 0
loss1/ave_pool (pooling)	kernel: 5	kernel: 6
loss1/fc (fully connected)	neurons: 1024	layer removed
loss1/classifier (output int. 1)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)
loss2/ave_pool (pooling)	kernel: 5	kernel: 6
loss2/fc (fully connected)	neurons: 1024)	layer removed
loss2/classifier (output int. 2)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)
loss3/classifier (output)	neurons: 1000 (ImageNet classes)	neurons: 50 (CORe50 classes)

C Hyperparameter Values for CORe50

Table 4: the hyperparameter values used for CaffeNet and GoogLeNet on CORe50. The selection was performed on run 1, and hyperparameters were then fixed for runs $2, \ldots, 10$.

Cumulative		
Parameters	*CaffeNet*	*GoogLeNet*
epochs, η (learn. rate)	4, 0.0025	4, 0.0025

Naive		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.005
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003
LWF		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Expanding	Expanding
map	[0.66...0.9] $\rightarrow$ [0.45...0.85]	[0.66...0.9] $\rightarrow$ [0.45...0.85]
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0002	2, 0.0002
EWC		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
max_F	0.001	0.001
λ	5.0e7	3.4e7
B_1: epochs, η (learn. rate)	2, 0.001	4, 0.002
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.000025	2, 0.000035
SI		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
ξ	1e-7	1e-7
$w_1, w_i(i > 1)$	0.00001, 0.005	0.00001, 0.005
max_F	0.001	0.001
λ	5.0e7	3.4e7
B_1: epochs, η (learn. rate)	2, 0.001	4, 0.002
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.00002	2, 0.000035
CWR		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
$w_1, w_i(i > 1)$	1.25, 1	1, 1
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003
CWR+		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003
AR1		
Parameters	*CaffeNet*	*GoogLeNet*
Head (see App. A)	Maximal	Maximal
ξ	1e-7	1e-7
$w_1, w_i(i > 1)$	0.0015, 0.0015	0.0015, 0.0015
max_F	0.001	0.001
λ	8.0e5	8.0e5
B_1: epochs, η (learn. rate)	2, 0.0003	4, 0.0003
$B_i, i > 1$: epochs, η (learn. rate)	2, 0.0003	2, 0.0003

D Hyperparameter Values for iCIFAR-100

Table 5: the hyperparameter values used for CifarNet [33] on iCIFAR-100. The selection was performed on run 1, and hyperparameters were then fixed for runs $2, \ldots, 10$.

Cumulative	
Parameters	*CifarNet*
epochs, η (learn. rate)	180, 0.005
Naive	
Parameters	*CifarNet*
Head (see App. A)	Maximal
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001
LWF	
Parameters	*CifarNet*
Head (see App. A)	Expanding
map	[0.5...0.9] $\rightarrow$ [0.45...0.85]
B_1: epochs, η (learn. rate)	20, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	20, 0.001
EWC	
Parameters	*CifarNet*
Head (see App. A)	Maximal
max_F	0.001
λ	8.0e7
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	25, 0.00002
SI	
Parameters	*CifarNet*
Head (see App. A)	Maximal
ξ	1e-7
$w_1, w_i(i > 1)$	0.00001, 0.00175
max_F	0.001
λ	6.0e7
B_1: epochs, η (learn. rate)	60, 0.0005
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.00002
CWR	
Parameters	*CifarNet*
Head (see App. A)	Maximal
$w_1, w_i(i > 1)$	1, 1
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001
CWR+	
Parameters	*CifarNet*
Head (see App. A)	Maximal
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001

AR1	
Parameters	*CifarNet*
Head (see App. A)	Maximal
ξ	1e-7
$w_1, w_i (i > 1)$	0.00015, 0.000005
max_F	0.001
λ	4.0e5
B_1: epochs, η (learn. rate)	60, 0.001
$B_i, i > 1$: epochs, η (learn. rate)	60, 0.001

E On Initializing Output Weight to Zero

It is well known that neural network weights cannot be initialized to 0, because this would cause intermediate neuron activations to be 0, thus nullifying backpropagation effects. While this is certainly true for intermediate level weights, it is not the case for the output level.

More formally, let θ_{ab} be a weight of level l, connecting neuron a at level $l-1$ with neuron b at level l and let net_x and out_x be the activation of neuron x before and after the application of the activation function, respectively; then, the gradient descent weight update is proportional to:

$$\frac{\partial L_{cross}(\hat{y}, t)}{\partial \theta_{ab}} = \frac{\partial L_{cross}(\hat{y}, t)}{\partial net_b} \cdot \frac{\partial net_b}{\partial \theta ab} = \frac{\partial L_{cross}(\hat{y}, t)}{\partial net_b} \cdot out_a \tag{12}$$

It is well evident that if out_a is 0, weight update cannot take place; therefore weights of levels up to $l-1$ cannot be all initialized to 0. For the last level (i.e., l coincides with output level), in case of softmax activation and cross-entropy loss, eq. 12 becomes [30]:

$$\frac{\partial L_{cross}(\hat{y}, t)}{\partial \theta_{ab}} = (\hat{y}_b - t_b) \cdot out_a = (out_a - t_b) \cdot out_a \tag{13}$$

and initializing θ_{ab} to 0 does not prevent the weight update to take place.